Article:

M. Möstl, A. Dörflinger, M. Albers, H. Michalik and R. Ernst, "Self-Adaptation for Availability in CPU-FPGA Systems Under Soft Errors," 2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2019, pp. 9-16.

DOI: 10.1109/AHS.2019.000-6

Self-Adaptation for Availability in CPU-FPGA Systems under Soft Errors

Mischa Möstl, Alexander Dörflinger, Mark Albers, Harald Michalik, Rolf Ernst

Institute of Computer and Network Engineering (IDA)
Technische Universität Braunschweig
Braunschweig, Germany
{moestl, doerflinger, albers, michalik, ernst}@ida.ing.tu-bs.de

***Abstract*—We introduce a model-based reliability estimation to preserve application availability in CPU-FPGA systems exposed to soft errors under varying environment conditions. The estimation is used as an in-system method to select a suitable configuration for changing radiation conditions. This allows systems to autonomously adapt their configuration in order to balance between reliability and performance. Such a self-adaptation goes beyond the state-of-the-art, where adaptation relies on preplanned reactive mode changes. By autonomously evaluating new configurations, our self-adaptation process is capable of increasing the availability by selecting the configuration with the desired application reliabilities for the current environment conditions.**

***Index Terms*—Autonomous Systems, Reliability, Self-adaption**

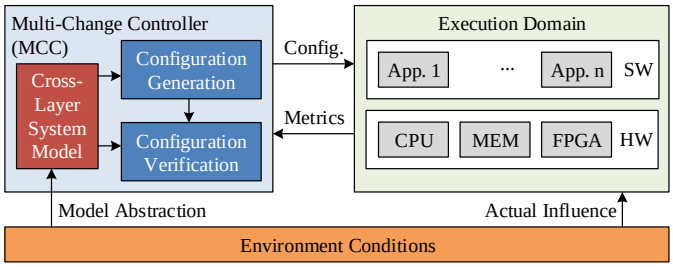

Fig. 1. Autonomous system setup, consisting of a model domain (blue, left) responsible for configuration and control, and an execution domain (green, right) consisting of the actual hardware and software.

I. Introduction and Motivation

Long life times of systems such as space probes often have the effect that the environmental conditions change, in which a system operates. Among the more common factors such as temperature and humidity is also radiation, which can cause Single Event Effects (SEEs) in electronic circuits and can ultimately lead to application failure. This is becoming a major concern with ever shrinking transistor size. To ensure a suitable reliability, traditional designs pursue an approach, where designers consider worst-case conditions under which certain applications of a system are expected to operate without failure, e.g. a solar flare for space applications. However, this often results in poor resource usage when operating in average conditions. One alternative is to equip applications with different modes of operation, to react to different environment conditions, e.g. to switch between redundant and non-redundant operation to maintain the availability of an application. Yet, such reactive responses to environment change imply that they are planned in advance, as different modes require different amounts of resources. Further, predefining reactions requires knowledge to what extent the environment is possibly changing, in order to design these modes of operation. Depending on the range of possible changes, this already opens up a huge design space. Although, automatic mechanisms are available to explore such a design space (e.g. [1]) in order to look for a balanced trade-off between fault-tolerance measures and performance, they still suffer from the fact that the operation points, e.g. the radiation level to expect, are still unknown. This makes it a hard challenge to design for high availability under varying conditions, as the reliability of an application cannot suitably adjust to environment conditions.

Further, both approaches, worst-case design and multiple operation modes with different fault-tolerance mechanisms, also suffer from the fact that they have to be repeated by the designer in the lab, e.g. when an application is added or removed from the system, or just updated. If the system hosts an application set with mixed-critical requirements, i.e. different safety levels, the situation becomes even more complicated. In mixed-criticality (MC) systems the designers have to guarantee that not only the applications operate with a sufficient reliability and performance, but also that applications of different criticalities are sufficiently independent of each other [2]. This motivates to automate the process of augmenting the applications with fault-tolerance mechanisms and to decide on whether the achieved reliability is sufficient for defined operation conditions. By outsourcing this process from the lab-based design phase to the operation phase in a system's lifetime, it allows the system to autonomously adapt to changing conditions.

Consequently, a two-step process follows: First, new configurations must be generated, which can be optimized according to an objective function. In a second step, the new configuration must be checked whether it fulfills all requirements of all applications in the system.

Figure 1 shows the conceptual setup of such a system.

https://doi.org/10.24355/dbbs.084-202105041454-0

A Multi-Change Controller (MCC) hosts the model domain, consisting of the model itself (red), as well as configuration generation and verification (blue). Based on a cross-layer model that captures relevant aspects of the system, new configurations can be generated and subsequently checked for requirements satisfaction. If a new configuration satisfies all requirements and is rated as an improvement to the current one, it can be deployed into the execution domain. Verification is separated due to the fact that not all requirements can be systematically considered during configuration generation. E.g. software response times are hard to optimize if arbitrary activation patterns are assumed. Consequently, an autonomous configuration and verification goes beyond a multi-dimensional optimization of requirement satisfaction. However, here we focus on application reliability, which is challenged by soft errors in the underlying HW platform.

Contribution: In this paper, we demonstrate a self-adaptive, model-based reconfiguration control that allows to adapt the system to varying environment conditions. Through adapting the configuration of the system to maintain a desired application reliability under the changing radiation conditions we increase application availability.

II. Related Work

The field of increasing application resilience against soft errors has a long-standing history in integrated circuit (IC) design. W.r.t. related work we want to focus on three core aspects of our paper: i) the overhead of protection mechanisms ii) the adaptivity of the system after deployment, i.e. after a lab-based design and integration phase, and iii) the system's self-awareness of its operating environment. Work such as [3], [1] allows to evaluate the overhead of specific protection mechanisms, w.r.t. performance, area, power, etc. However, they only allow a lab-based design space exploration, and only deliver results for fixed operating conditions. For entirely software-based applications, e.g. [4] provides a multi-dimensional optimization, which includes reliability, to exploit the potential of network on chip based MPSoCs. The described optimization performs application mapping based on resource slack, i.e. unused resources. The remapping can also be applied in-field, e.g. after one resource fails. In case of sufficient slack in this situation, a new application mapping can be determined. On the other hand, [5] describes an FPGA hosted satellite application that is able to reactively switch its mode of operation to increase its reliability. The predetermined configurations exhibit different speedup factors and failure probabilities. Switching is performed based on the observed application failure rate. [6] extends this approach by using an internal Block RAM as a radiation particle sensor in order to be independent of application failure detection for suitable reactions.

Our approach of in-field adaptation that reasons on a model-based representation of the system is new compared to the state-of-the-art, in the sense that it aims to replace lab-based application integration. In doing so, it allows the system to adapt its applications to different environment conditions. I.e. through observing its own environment and the health state of its applications, the system becomes self-aware [7], which goes beyond the reactive control e.g. discussed in [5], [6]. W.r.t. self-awareness approaches for systems, we extend the state of art (e.g. summarized in [8]) by providing a platform centric method. More precisely, the self-awareness is part of the system and its resource management rather than being application specific. Another attempt in this direction are self-aware middleware solutions such as [9]. However, the MCC solution proposed by us does not need to be deployed as a run-time mechanism within the system. To our knowledge, it is the first self-aware approach that allows the platform to assess the reliability of its applications and adapt its configuration based on the expected environment. In doing so, it ensures the availability of critical applications with an expected reliability. Without the adaptation the reliability would, in harsher environments, decrease below the required one, and in consequence limit the availability of the application.

III. Application Reliability Modeling

In today's design processes the reliability under soft errors is rarely determined on application basis, but rather a property of the enabling HW-SW platform. In consequence, the availability of individual applications under soft errors is thus only anticipated based on the reliability estimation of the platform.

However, due to the high integration level of today's ICs it is required to consider reliability on hierarchical levels. A System-on-Chip (SoC) can be considered as a system of systems, as it is composed of multiple components with dedicated failure rates. This applies for CPU-FPGA SoCs in particular, which posses a differing soft error susceptibility of processor cores and FPGA fabric. Furthermore, when running applications with mixed criticality on the same hardware components, the reliability and consequently availability goals differ. Typically, a highly critical application requires a higher success rate compared to a less critical application. These aspects require analyzing reliability on a decomposed resource level and for different applications separately.

In the following we will elaborate a cross-layer model for a reliability representation of computation systems, which satisfies system decomposition. The sub-components of a CPU-FPGA SoC feature different fault susceptibilities, therefore reliability is affected when migrating the execution of tasks from software to hardware and vice versa. We allow multiple tasks to be mapped to a sub-component (e.g. an FPGA region or CPU core) which implies time-sharing. For the FPGA this means that it supports Dynamic Partial Reconfiguration (DPR) for switching between hardware tasks in reconfigurable FPGA regions. On a CPU core, time-sharing of software tasks is carried out by context switch.

The reliability prediction stage (Figure 2) is part of the configuration verification process in Figure 1 and validates the system configuration candidates against reliability requirements. In a first step, the model estimates task failure rates. For this, the susceptibility of resources contained in the HW platform has to be characterized. Tasks need to be analyzed

https://doi.org/10.24355/dbbs.084-202105041454-0

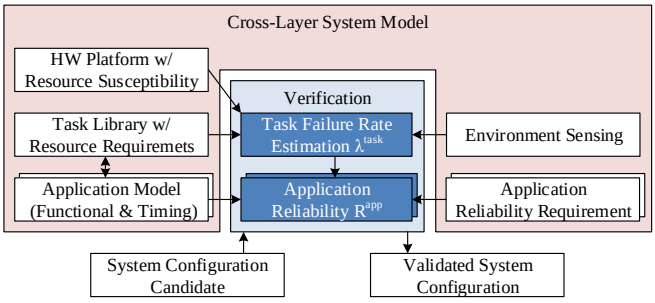

Fig. 2. MCC Reliability Prediction Stage.

respectively to the resources being occupied during their execution. Furthermore, the environment of operation (radiation particle flux, temperature, etc.) has to be measured or provided. A detailed discussion of the task failure rate estimation follows in subsection III-A.

In a second step, the model allows to compute the reliability for each application of the system. An application is described in a functional- and timing model, which provide all relevant information for determining the reliability from task failure rates. A detailed discussion of the application reliability calculation follows in subsection III-B.

A. Task Failure Rate Estimation

Calculating an application reliability from error rates on bit level quickly becomes a sophisticated problem for complex systems. The RAP model in [10] suggests computing the probability of a bit error for different fault origins in a first step. Subsequently, the probability of application failures is calculated by applying transformation functions between multiple abstraction levels, e.g., bit errors are transformed into word errors, which are transformed in interface errors, and so forth. This in-depth approach generates very accurate results, if 1) reliable data exists on bit errors, and if 2) the transformations between abstraction levels are known. This typically applies for the FPGA fabric of CPU-FPGA SoCs: bit error probabilities can be calculated from reliability reports such as [11] and transformation functions can be easily generated, e.g., by running fault injection tests such as [12]. However, the RAP model has its limitations for the CPU part of many SoCs due to imprecise data on bit error rates and an incomplete description of the system through transformation functions. Typically, fault injection tools are not available for the processor systems, and beam tests are too expensive to generate transformation functions for specific applications. To overcome this problem for practical applications, we suggest analyzing failure rates on different decomposition levels. An error rate analysis on bit level produces the most accurate results and is aspired, but not always possible. Here we fall back to an incomplete decomposition, which is compensated through higher conservative upper bounds on error probabilities. The decomposition of resources ends if there are no error rates or transformation functions available for a deeper level. For the Xilinx Zynq-7000 SoC family, being used in the case study in section V, the FPGA fabric can be decomposed down to bit level and comprises configuration memory (CMEM), registers (flip-flops, latches), logic (LUTs), and block memory (BRAM) respectively. The decomposition of the ARM processor system ends on ALU, FPU, L1 and L2 caches.

A soft error in one of the named resources may (but not always will) result in a failure in the task currently executed. This is expressed in a vulnerability factor v. Hence, the task failure rate calculates as in Equation 1, similar to the parts count analysis [13]. For each resource type i being occupied by a task during its execution, the product of quantity N_i, failure rate λ_i, and vulnerability v_i contributes to the task failure rate.

$$\lambda^{task} = \sum_{i=1}^{n} N_i \lambda_i v_i \tag{1}$$

The soft error failure rate λ_i for each resource type of the hardware platform is computed as a function of current environment conditions. As an example, the failure rate induced by a particle flux Φ can be computed from the resource's cross section (Equation 2). The calculation considers the energy distribution of particles by using a cross section fitted with Weibull parameters. For simplicity in the case study in section V, we estimate an upper bound of the upset rate by multiplying the saturating cross section σ_{sat} with the total particle flux Φ.

$$\lambda_i = \int_0^\infty \sigma_i(E)\Phi(E)dE < \sigma_{i,sat} \cdot \Phi \tag{2}$$

The particle flux in a changing environment can be measured by on-board sensors (e.g. [6]) or be supplied from external information sources (e.g. NASA's Space Weather Prediction Center[1]).

When considering SEEs as the main fault origin, the impact of each resource type on the task failure rate can be calculated using Equation 1 and 2. The following parameters have to be determined as inputs:

- Susceptibility of resources to SEEs (cross section σ_i) as a parameter of the hardware platform.
- Number of resources occupied by a task (N_i) and
- probability of a soft error to result in a failure (vulnerability factor v_i) as parameters of the resource requirement library.
- Radiation particle flux (Φ) provided through an environment sensing.

Those parameters will be discussed in detail for the resource types being present in a CPU-FPGA coupled system, such as a Xilinx Zynq-7000 SoC.

1) FPGA fabric resources: Most non-destructive SEEs that disturb the FPGA fabric eventually corrupt the content of a storage element. There are three different memory types with distinctive soft error behavior present in the FPGA fabric.

Configuration Memory (CMEM): The CMEM defines the

[1] swpc.noaa.gov

https://doi.org/10.24355/dbbs.084-202105041454-0

logical operation of the FPGA. It sets the combinatorial and sequential circuitry of functional blocks and their connectivity. The cross section σ_{CMEM} per bit can be computed from failure rates being provided in reliability reports [11] or beam tests [14]. The number of bits N_{CMEM} used by a task scales linearly with the occupied FPGA area, i.e. the size of a reconfigurable region. Not all CMEM cells affect the proper operation of a specific design, hence only a part of the CMEM has to be considered for reliability. This will be expressed by weighting the CMEM with a vulnerability factor v_{CMEM}, which denotes the proportion of *essential* bits. Soft errors in *non-essential* bits do not lead to a function failure and can be safely ignored. [12] and [15] characterize the failures of specific FPGA designs by classifying deviations observed in the output vectors after injecting errors into the CMEM. This allows an accurate calculation of the proportion of essential bits of a design. However, such a detailed characterization is costly and not available for all FPGA designs. Alternatively, a conservative estimate of essential bits is being reported by FPGA vendors tools.

Registers: Sequential circuits of an FPGA design use flip-flops, latches, and distributed RAM as storage elements. The cross section of a register σ_{reg} is assumed to be similar to the cross section of a CMEM bit, by reason of the similar structure in silicon. The number of registers N_{reg} being used in an FPGA design is reported in the respective toolchains. For the reliability prediction, it will be assumed that all registers are essential for correct operation (vulnerability factor $v_{reg} = 1$). Note that this is a conservative over-estimation, as a design might contain surplus registers that are not addressed in normal operation. When applying redundancy schemes, e.g. Triple Modular Redundancy (TMR), only multiple soft errors result in a function failure. Such techniques decrease the vulnerability factor by order of magnitude, however they require a high resource overhead.

Block Random Access Memory (BRAM): A typical FPGA design requires memory blocks to buffer input- and output data and to store intermediate results. In order to serve these needs, the FPGA fabric contains BRAM resources. The regular structure of BRAM is more susceptible to soft errors than CMEM and registers, hence [14] reports a higher cross section σ_{BRAM}. The vendor's synthesis tools directly generate the number N_{BRAM} being instantiated for a task. Typically, not all bits of a BRAM are actively used during operation of the task, which is expressed by a vulnerability factor $v_{BRAM} < 1$. Error Detection and Correction (EDAC) codes can mitigate soft errors in BRAM, resulting in a further decreased vulnerability factor v_{BRAM}.

2) Processor resources: Analogous to the FPGA fabric, different resources can be distinguished within a processor system. For the investigated Zynq-7000 architecture, the SEE sensitivity and respective cross sections σ_i have been evaluated for five functional blocks (on-chip memory, L1d cache, ALU, FPU, and peripheral) [16]. At this point we resign a further decomposition of resources due to the lack of test data on bit error susceptibility and unknown transformation between bit error rates and application error rates in the Cortex-A9 system. The resources can be either used ($N_i = 1$) or not used ($N_i = 0$), e.g., when compiling a task without FPU features or disabling caches. As there is no further data available on the vulnerability of dedicated assembly instructions, a conservative vulnerability of $v_{ALU/FPU} = 1$ is assumed. Mitigation of soft errors in processor systems, such as lockstep or TMR techniques [17], can be modeled by a decreased vulnerability factor, however none of these features are available in the investigated Cortex-A9 architecture. For caches, the vulnerability scales with the cache hit rate, because each cache miss masks a soft error and the erroneous bit will be replaced. Note that this approximation also accounts for bit flips in the cache logic, since the control logic is also considered while determining the cross section.

3) Memory: Data being stored in external memory is also exposed to soft errors. Particularly memories targeted for space applications have been thoroughly tested and memory cell cross sections σ_{mem} are provided [18]. This typically allows a contemplation of memory errors on bit level. The number of memory bits used by a task N_{mem} scales with program- and data memory size. The vulnerability is conservatively estimated with $v_{mem} = 1$, which implies that each memory word will be used during task execution eventually. Again the vulnerability can be reduced through soft error mitigation techniques such as EDAC codes for memory content protection.

B. Computing Application Reliability

Up to this point, we have characterized the susceptibility of HW resources and calculated task failure rates. However, in order to predict the reliability also the temporal aspect is of importance, i.e. how long a task occupies the HW and is thus susceptible to SEEs. In order to be able to predict application reliability, applications and platform are described by a cross-layer model. In this model, applications are decomposed in a functional model, which is subsequently mapped to task model that describes temporal behavior of the application, i.e. how long and often HW resources are occupied.

Definition 1. A *functional model* is a graph $\mathcal{FG} = (F, L, \xrightarrow{w}, \xrightarrow{r})$ with F as the set of Function Blocks (FBs), L denoting the set of data labels and $\xrightarrow{w}, \xrightarrow{r}$ defining two precedence relations between FBs and labels and vice versa denoting writing and reading of labels.

This is similar to a control and data flow graph, with the exceptions that exchanged data is explicitly modeled by labels and the actual control flow may jump and branch within a FB. In order to be aware of timing implications, the FBs of $\mathcal{FG}$ are mapped to executable tasks of the timing model. Labels are mapped to storage resources.

Definition 2. A *timing model* is a tuple $\mathcal{TG} = (\mathcal{T}, \mathcal{R}, \rightarrow, \xi)$ with $\mathcal{T}$ representing the set of tasks, $\rightarrow$ defining a directed precedence relation between tasks such that output/completion events become activation events of the succeeding one, $\mathcal{R}$ as the set of HW resources, and ξ as a function that maps each task to the resources that provide execution time to its jobs

https://doi.org/10.24355/dbbs.084-202105041454-0

according to the resource's scheduling policy.

Note that tasks can either be implemented as software tasks, executed on processor resources, or hardware tasks, executed in a dynamically reconfigurable region of the FPGA. The concrete execution of a task is referred to as a job, i.e. if a software task is activated twice, two jobs must be processed. Multiple tasks can be mapped to the same resource, implying that the resource is shared in time. Further, one task can spawn jobs on more than one resource if it is mapped to many. Resource sharing consequently implies, that a schedule is maintained for each resource in $\mathcal{R}$. Through the mapping of FBs to tasks, and again tasks to resources, an execution trace can be generated from the schedules. The trace consists of the necessary jobs of all tasks to realize the functionality described by $\mathcal{FG}$.

Again, the resources are described by a hierarchical resource model, i.e. the region of an FPGA is characterized by the maximum amount of BRAM, CMEM, etc. Similarly, processor resources are decomposed as introduced in the previous subsection.

An application *reliability requirement* is expressed as a success probability $p_i \in [0;1]$ of one execution of a connected subgraph G_{req} of the functional model $\mathcal{FG}$, or as probability of failure per hour (PFH) / probability of failure on demand (PFD) requirements as e.g. mandated by safety standards such as IEC 61508 [2].

The failure rate is assumed to be constant during the execution of one job, hence the reliability of one job is computed by:

$$R^{job} = \exp\Big(- \lambda^{task} \cdot T_{vul}\Big) \tag{3}$$

whereas λ_{task} is computed according to Equation 2. T_{vul} is the time a task's job is actively occupying the resources. In our approach, we determine T_{vul} for each necessary job of a task individually. Therefore, the mapping of each FB and data label must be known as well as the time the task is vulnerable, i.e. is active on the resource. For jobs of a HW task this implies the time of the execution plus the preceding configuration, if the HW task needs to be loaded, i.e. there was no directly preceding job. For software tasks we assume, that its vulnerable time T_{vul} is the maximum time a job of the task is active, i.e. the vulnerable time is the worst-case response time (WCRT) of the task. The response time is defined as the time interval between activation of a job and its subsequent termination. It includes the time when a job is e.g. preempted due to higher priority tasks executing. With this over approximation, we make certain that we require the HW to be fault free as long as state of a job is maintained in its registers. In our setup, the WCRT of a task is determined based on Compositional Performance Analysis (CPA) [19], which processes the timing model.

The success tree of any reliability constraint can be treated as a line topology. Although the functional graph $\mathcal{FG}$ might contain parallel strands, the successful execution still depends on the successful execution of all the involved jobs of all involved tasks. We assume that the reliability of the individual jobs is statistically independent, as we conservatively assume that no fault-masking occurs due to burst errors. Consequently the application reliability R^{app} of a connected subtree of $\mathcal{FG}$ can be calculated in combination with Equation 3 as the product of all involved jobs derived from the schedules in $\mathcal{TG}$:

$$R^{app} = \prod_{j=1}^{m} R_j^{job} \tag{4}$$

All the assumptions made for Equation 3 and 4 are conservative. For software tasks, we only take into account conservative upper bounds for the WCRT, as well as for hardware tasks, where each task is treated as fully vulnerable, even if a module is not yet fully configured.

IV. Self-Adaptation

In order to adapt the system to changing conditions, the MCC performs a model-based integration of available function implementations. In this integration process, it tries to find a configuration that matches the input models and their requirements. In general, the trigger for the MCC to change the configuration of the system, are any changes to the input models of the MCC. Yet, in this paper we specifically focus on the environment model of the system.

In general, the MCC enriches model layers and synthesizes new layers in a stepwise process, starting from the top-most layer. Depending on the system's hardware and software setup, i.e. number of CPU cores and FPGAs, the operating system and its model of computation, various layers can be generated. The top-most layer is implementation agnostic, i.e. it only specifies the intended functionality without dictating whether a function block is implemented in either software or hardware. In the scope of this paper, we start the integration from the functional layer $\mathcal{FG}$ and derive a configuration of the system that is specified by the task layer and resource layer of the cross-layer model.

The synthesis of a configuration consists of a series of three distinct step types: *parameter decision*, *transformation*, and *check*. First, parameter decisions are preformed on the functional model, before a transformation into tasks and resources can be conducted. Typical *parameter decisions* are e.g. mapping decisions of functions to processing units (FPGA or CPU), the selection of the redundancy mode of a task (TMR or none) or the synthesis of a schedule. The parameter decisions typically prepare the second step type, *transformation*: E.g. selecting a hardware implementation where data labels are pipelined over an interconnect between hardware components (cf. Figure 3) instead of exchanging it via the main DRAM-memory. The third step type, *check*, is an admission test for a specific layer. It performs validation tests on the transformation, e.g. whether the transformation meets the available hardware resources' capabilities such as available field-programmable gate array (FPGA) area. If a check can not be passed successfully, the process rolls back transformations and design decisions, and iterates over another variant.

https://doi.org/10.24355/dbbs.084-202105041454-0

For component-based operating systems, detailed account on parameter decisions and transformations is given in [20]. Since this process might lead to an exhaustive design-space exploration, we apply heuristics where possible to mimic a human designer in a manual integration process. For the case study presented in section V, an optimization algorithm generates configurations which are optimized for throughput of the presented application. However, note that not all requirements of all applications can be holistically optimized, hence the third step type *check* is necessary. A particular admission test is the reliability estimation for each application as described in the previous section.

The change process of the MCC, i.e. the search for new configurations, in the scope of this paper is triggered by a change in particle flux, and respectively by a change in the environment model. Through constant self-adaption of the system, its availability is increased, i.e. the fraction of time in which all minimum requirements are at least fulfilled. Note that this search can execute as a background task and does not necessarily need to be conducted once a requirement is already violated. Furthermore, the MCC can be used for other self-adaptations as well. In this case, changes in other input models may trigger the MCC. For instance, a change of the functional layer could request other or additional functionalities to be executed on the platform.

V. Case-Study and Discussion

For an evaluation of the self-adaption and reliability prediction mechanism of the MCC, a stereo-vision application will be analyzed. In the given application, a set of left and right images is being processed according to the functional graph depicted in the upper half of Figure 3. Note that besides the stereo-vision application shown in the $\mathcal{FG}$ further applications can share the platform. Both images are converted from bayer to RGB format (*Debayer*) and subsequently the *Rectify* FB removes camera distortions. *Stereomatch* computes a disparity map, which is converted to a pointcloud (*Disp2Pc*), and finally filtered in a pass through *Filter*. For each FB an implementation in both hardware and software is available, except for the pass through filter for which only a software implementation exists. Redundant hardware implementations using TMR are currently being developed. Hardware tasks are able to time-share the available FPGA resources and are scheduled in predefined reconfigurable regions of the FPGA fabric using DPR. The implementation variants of the tasks differ in their failure rate and execution time. By choosing different mappings from FBs to either standard or TMR tasks, multiple configurations with varying reliability and performance can be generated. A lower performance (or higher makespan) implies a lower stereo-vision frame rate and possible robot speed, hence we do not force a hard requirement on this parameter.

The MCC reliability prediction requires a parametrization of the resource susceptibilities of the hardware platform. This is expressed as cross sections σ_i and has been retrieved from [14] for the FPGA resources CMEM, registers, and BRAM. Respective values for processor resources ALU, FPU, and L1

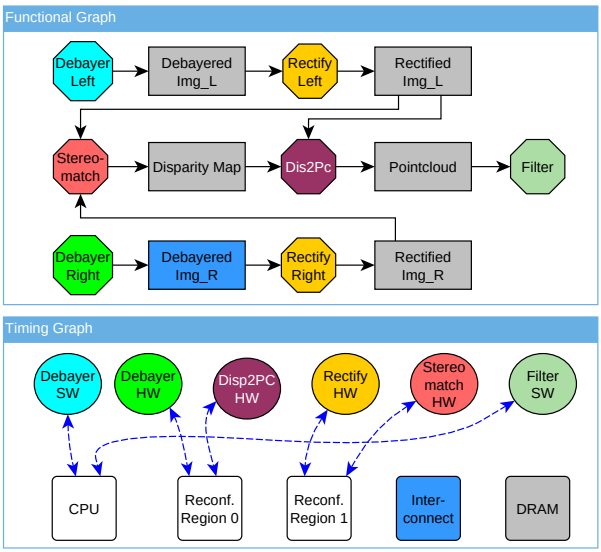

Fig. 3. The cross-layer model for one configuration of the case study application: The upper half shows the $\mathcal{FG}$ and the bottom the $\mathcal{TG}$. Mappings of tasks to resources are depicted by blue arrows, while the mapping from $\mathcal{FG}$ to $\mathcal{TG}$ is indicated by color, i.e. identical color denotes a mapping.

caches have been retrieved from [16]. The L2 cross section has been extrapolated from σ_{L1}, assuming an identical cell layout for L1 and L2.

Furthermore, the model requests the resource requirements of the task library. The number of resources occupied by hardware tasks has been obtained from resource usage reports of the Vivado toolchain. Reports on essential bits of the CMEM are used to compute precise vulnerability factors v_{CMEM}. Register and BRAM vulnerabilities have not been analyzed in detail yet, hence a conservative values $v_{reg/BRAM} = 1$ are assumed. Respective model inputs for TMR hardware tasks are extrapolated from their non-TMR variant. Software tasks have been compiled with FPU flags and all caches enabled, therefore all processor resources ALU, FPU, L1d, L1i, and L2 are being used. The cache hit rate has been obtained for each software task for all caches through cachegrind ([21], valgrind.org), and allows to provide an accurate estimation of cache vulnerability factors.

A preceding step of the MCC generates configuration candidates for the stereo-vision application with varying compositions of (TMR-) HW and SW tasks. This step generates solutions by exploring the design space through iterative assignement of the implementation variant, e.g. whether TMR is enabled or not, and mapping to execution resources in $\mathcal{R}$. The performance is optimized by minimizing the makespan of the functional graph using an algorithm developed in [22]. Non-reasonable solutions of the full permutation of tasks for the given solution are rejected, leaving 128 candidates to be analyzed in the here discussed reliability prediction stage.

Figure 4 plots the results of the reliability estimation as PFH over makespan for three different particle flux intensities. The largest flux Φ_3 being analyzed roughly corresponds to an environment in a geosynchronous earth orbit.

https://doi.org/10.24355/dbbs.084-202105041454-0

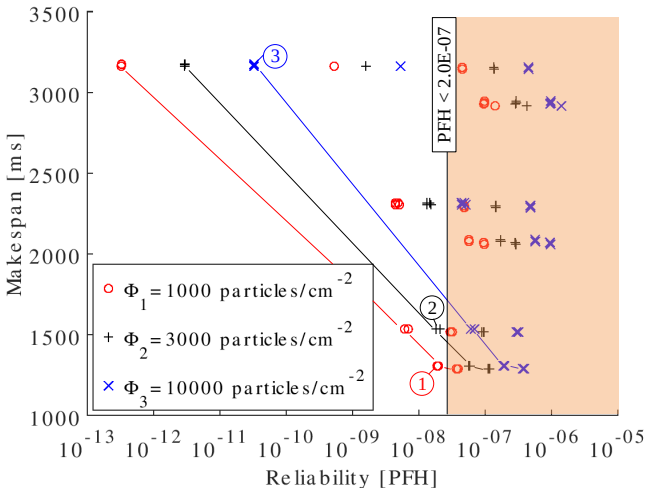

Fig. 4. Trade-off between reliability and makespan.

Note that some of the 128 configuration candidates resemble each other, leaving 13 clusters in the solution space. A multi-objective optimization on both makespan and PFH would reject all solutions above a pareto front, which leaves 4 contemplable clusters per flux intensity.

For an initial or updated configuration, the reliability prediction stage checks the fulfillment of all reliability requirements. Once a valid configuration has been found, it will be deployed into the execution domain. E.g., the discussed application may start during a low-radiation time period (Φ_1) with a high-performance configuration and a makespan of 1306 ms (marked as ① in Figure 4). During operation, the reliability requirements are reevaluated on any environment change. Once any reliability requirement check fails, new configuration candidates are requested until a valid solution has been found. Subsequently, the new configuration is deployed. For instance, once the radiation increases to Φ_2, the given reliability requirement is violated as the configuration moves into the forbidden red zone of Figure 4. Hence, alternative configurations are requested and checked, until the system settles for a configuration marked as ② in Figure 4. The makespan increases to 1534 ms, but the reliability requirement is met. Hence, the system stays available. For even higher particle flux values (e.g. Φ_3), a configuration marked as ③ with a makespan of 3162 ms will be selected.

While the runtime of the configuration candidate generation scales with the complexity of the $\mathcal{FG}$ that has to be mapped, the computation of reliability predictions is swift. Although the predictor has to extract job dependencies from $\mathcal{TG}$, which in theory is an exponential problem, practical instances compute within $\approx$ 1s on standard desktop computers[2].

VI. Conclusion

We have presented two contributions: First, we have demonstrated the applicability of our model and approach to a complex system structure, consisting of a CPU-FPGA SoC as platform, and a sophisticated software application mapped to it. Second, we modeled and implemented a reliability prediction within a reconfiguration framework that allows a system self-adaptation based on the awareness of particle flux in its environment. Therefore, we introduced a task failure rate estimation based on different decomposition levels. The calculation for hardware tasks mostly relies on FPGA vendors' toolchain reports and therefore can be performed easily for practical applications. We envision that predicting the reliability under changing conditions and triggering timely reconfigurations for e.g. safety critical applications will lead to a more economic resource usage in benign environments, yet ensuring availability and reliability constraints. However, the model parameters, particularly for resource cross sections, need to be reasonably accurate for in-field deployment for such an approach. Further radiation testing (in particular of the CPU systems) would allow further decomposition of the system and hence more accurate reliability estimations. Also, applying the model to a stereo-vision application has shown, that reliability and performance are concurrent goals, especially if multiple applications have to be considered. Future work can further analyze their correlation by feeding the here proposed reliability prediction model with multiple redundancy variants (e.g., both spatial and temporal DMR / TMR).

[2]Core i5-3210, 8GiB RAM, singlethreaded

Acknowledgment

This work is part of the DFG Research Group FOR 1800 "Controlling Concurrent Change".

References

[1] P. v. Stralen and A. Pimentel, "A safe approach towards early design space exploration of fault-tolerant multimedia mpsocs," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '12. New York, NY, USA: ACM, 2012, pp. 393–402.

[2] *IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2nd ed., The International Electrotechnical Commission - IEC, April 2010.

[3] L. G. Szafaryn, B. H. Meyer, and K. Skadron, "Evaluating overheads of multibit soft-error protection in the processor core," *IEEE Micro*, vol. 33, no. 4, pp. 56–65, July 2013.

[4] B. H. Meyer, A. S. Hartman, and D. E. Thomas, "Cost-effective Lifetime and Yield Optimization for NoC-based MPSoCs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, no. 2, pp. 12:1–12:33, Mar. 2014. [Online]. Available: http://doi.acm.org/10.1145/2535575

[5] H. Zhang, M. A. Kochte, M. E. Imhof, L. Bauer, H. . Wunderlich, and J. Henkel, "Guard: Guaranteed reliability in dynamically reconfigurable systems," in *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014.

[6] R. Glein, B. Schmidt, F. Rittner, J. Teich, and D. Ziener, "A Self-Adaptive SEU Mitigation System for FPGAs with an Internal Block RAM Radiation Particle Sensor," in *IEEE 22nd Annual Intern. Symposium on Field-Programmable Custom Computing Machines*, May 2014.

[7] P. R. Lewis, M. Platzner, B. Rinner, J. Tørresen, and X. Yao, "Self-aware Computing: Introduction and Motivation," in *Self-aware Computing Systems: An Engineering Approach*, ser. Natural Computing Series, P. R. Lewis, M. Platzner, B. Rinner, J. Tørresen, and X. Yao, Eds. Cham: Springer International Publishing, 2016.

[8] S. Kounev, X. Zhu, J. O. Kephart, and M. Kwiatkowska, "Model-driven Algorithms and Architectures for Self-Aware Computing Systems (Dagstuhl Seminar 15041)," *Dagstuhl Reports*, vol. 5, no. 1, pp. 164–196, 2015. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2015/5038

[9] S. Sarma, N. Dutt, P. Gupta, N. Venkatasubramanian, and A. Nicolau, "CyberPhysical-System-On-Chip (CPSoC): A self-aware MPSoC paradigm with cross-layer virtual sensing and actuation," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2015.

[10] A. Herkersdorf *et al.*, "Resilience Articulation Point (RAP): Cross-layer dependability modeling for nanometer system-on-chip resilience," *Microelectronics Reliability*, vol. 54, no. 6, pp. 1066 – 1074, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026271413004630

[11] *Device Reliability Report*, v10.8.2 ed., Xilinx, Inc., 2018.

[12] H. Michel, H. Guzmán-Miranda, A. Dörflinger, H. Michalik, and M. A. Echanove, "SEU fault classification by fault injection for an FPGA in the space instrument SOPHI," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, July 2017.

[13] Department of Defense, "MIL-HDBK-217F Reliability Prediction of Electronic Equipment," Dec 1991.

[14] M. Amrbar, F. Irom, S. M. Guertin, and G. Allen, "Heavy ion single event effects measurements of Xilinx Zynq-7000 FPGA," in *2015 IEEE Radiation Effects Data Workshop (REDW)*, July 2015.

[15] J. Tonfat, L. Tambara, A. Santos, and F. Kastensmidt, "Method to analyze the susceptibility of HLS designs in SRAM-based FPGAs under soft errors," in *Applied Reconfigurable Computing*, V. Bonato, C. Bouganis, and M. Gorgon, Eds. Springer Intern. Publishing, 2016.

[16] W. Yang, X. Du, C. He, S. Shi, L. Cai, N. Hui, G. Guo, and C. Huang, "Microbeam heavy-ion single-event effect on Xilinx 28-nm system on chip," *IEEE Transactions on Nuclear Science*, vol. 65, no. 1, Jan 2018.

[17] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini, "Fault-tolerant platforms for automotive safety-critical applications," in *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, ser. CASES '03. New York, NY, USA: ACM, 2003.

[18] M. Herrmann, "Radiation characterization of highly integrated DDR3 SDRAM devices for spaceborne mass storage applications," Dissertation, TU Braunschweig, 2018.

[19] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System Level Performance Analysis - the SymTA/S Approach," in *IEE Proceedings Computers and Digital Techniques*, 2005.

[20] J. Schlatow, M. Möstl, R. Ernst, M. Nolte, I. Jatzkowski, and M. Maurer, "Towards model-based integration of component-based automotive software systems," in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, Oct. 2017, pp. 8425–8432.

[21] N. Nethercote, "Dynamic binary analysis and instrumentation," Dissertation, University of Cambridge, 2004.

[22] A. Dörflinger, M. Albers, J. Schlatow, B. Fiethe, H. Michalik, P. Keldenich, and S. Fekete, "Hardware and software task scheduling for ARM-FPGA platforms," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018.

https://doi.org/10.24355/dbbs.084-202105041454-0